UNDERSTANDING ADDITION IN TRANSFORMERS

Philip Quirke[†] **Fazl Barez**[†‡]
[†]Apart Research
[‡]Department of Engineering Sciences, University of Oxford
fazl@robots.ox.ac.uk

ABSTRACT

Understanding the inner workings of machine learning models like Transformers is vital for their safe and ethical use. This paper presents an in-depth analysis of a one-layer Transformer model trained for n-digit integer addition. We reveal that the model divides the task into parallel, digit-specific streams and employs distinct algorithms for different digit positions. Our study also finds that the model starts calculations late but executes them rapidly. A rare use case with high loss is identified and explained. Overall the model's algorithm is explained in detail. These findings are validated through rigorous testing and mathematical modeling, contributing to the broader works in *Mechanistic Interpretability*, *AI safety*, and *alignment*. Our approach[1] opens the door for analyzing more complex tasks and multi-layer Transformer models.

1 INTRODUCTION

Understanding the underlying mechanisms of machine learning models is essential for ensuring their safety and reliability (Barez et al., 2023; Olah et al., 2020b). Specifically, the sub-field of *mechanistic interpretability* within machine learning interpretability aims to dissect the behavior of individual neurons and their interconnections in neural networks (Räuker et al., 2022). This pursuit is part of a larger endeavor to make the decision-making processes of complex machine learning models transparent and understandable. Although models like Transformers have shown remarkable performance on a myriad of tasks, their complexity makes them challenging to interpret. Their multi-layered architecture and numerous parameters make it difficult to comprehend how they derive specific outputs (Vig, 2019). Further, while simple arithmetic tasks like integer addition may be trivial for humans, understanding how a machine learning model like a Transformer performs such an operation is far from straightforward.

In this work, we offer an in-depth analysis of a one-layer Transformer model performing n-digit integer addition. We show that the model separates the addition task into independent digit-specific streams of work, which are computed in parallel. Different algorithms are employed for predicting the first, middle, and last digits of the answer. The model's behavior is influenced by the compact nature of the task and the specific format in which the question is presented. Despite having the opportunity to begin calculations early, the model actually starts later. The calculations are performed in a time-dense manner, enabling the model to add two 5-digit numbers to produce a 6-digit answer in just 6 steps (See Fig. 1). A rare use case with high loss was predicted by analysis and proved to exist via experimentation. Our findings shed light on understanding and interpreting transformers. These insights may also have implications for AI safety and alignment.

Our results demonstrate the model's unique approach applies to integer addition across various digit lengths (Refer Appendixes B and C). Our theoretical framework provides a mathematical justification for the model's behavior, substantiating our empirical observations and offering a foundation for future work.

Our main **contributions** are:

[1]To encourage re-use and reproducibility we made out code available here:
https://github.com/apartresearch/Interger_Addition

- Reformulation of the traditional mathematical rules of addition into a framework more applicable to Transformers.
- Detailed explanation of the model's (low loss) implementation of the addition algorithm, including the problem and model constraints that informed the algorithm design.
- Identification of a rare use case where the model is not safe to use (has high loss), and explanation of the root cause.
- Demonstration of a successful approach to elucidating a model algorithm via rigorous analysis from first principles, detailed investigation of model training and prediction behaviours, with targeted experimentation, leading to deep understanding of the model.

Below, we provide an overview of related work (§3), discuss our methodology (§4), describe our mathematical framework (§5), our analysis of model training (§6) and model predictions (§7). We conclude with a summary of our findings and directions for future research (§8).

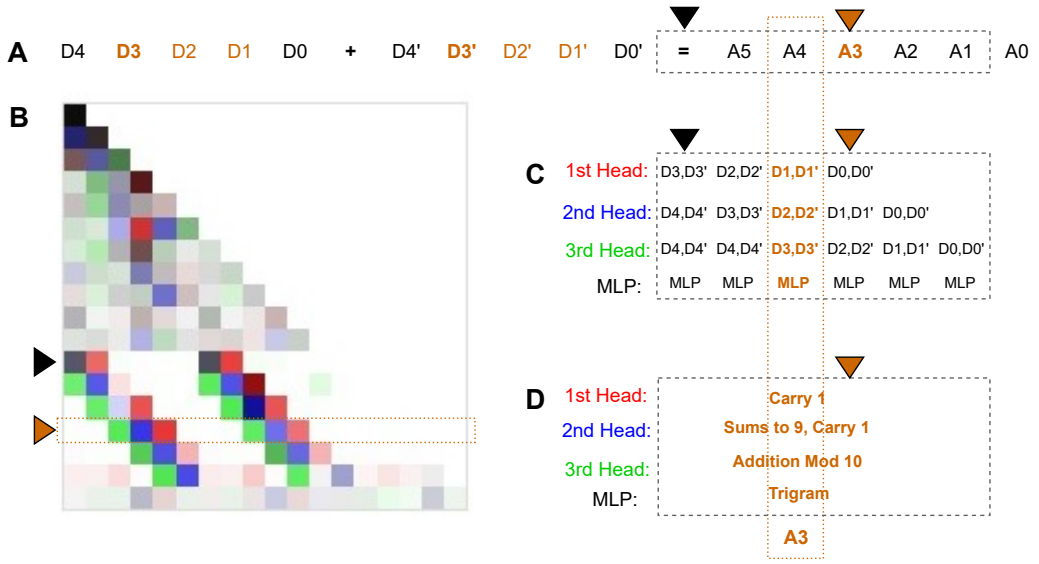

Figure 1: Illustration of the transformer model's attention pattern when adding two 5-digit integers. The model attends to digit pairs sequentially from left to right, resulting in a "double staircase" pattern across rows. **A:** The 5 digit question is revealed token by token. The "10s of thousands" digit is revealed first. **B:** From the "=" token, the model attention heads focus on successive pairs of digits, giving a "double staircase" attention pattern. **C:** The 3 heads are time-offset from each other by 1 token such that, in each row, data from 3 tokens is available. **D:** To calculate A3, the 3 heads do independent simple calculations on D3, D2 and D1. The results are combined by the MLP layer using trigrams. A3 is calculated one token before it is needed. This approach applies to all answer digits, with the first and last digits using slight variations of the approach.

2 Background

We focus on a single-layer transformer model with a vocabulary of size V containing the symbols "0" to "9", "+" and "=". The model converts the human readable input (e.g. "12345+67890=") into an input sequence $(\mathrm{x}_1, \ldots, \mathrm{x}_p)$ where each $\mathrm{x}_i \in \{1, \ldots, V\}$. Tokens are mapped to $\mathrm{d_e}$ dimensional embeddings by selecting the x_i-th column of $E \in \mathbb{R}^{d_e \times V}$. The model processes the input tokens, using a mechanism called "self-attention". Each input token is passed through a self-attention mechanism that calculates weighted relationships between all input tokens - capturing the importance of each token relative to others. The model then aggregates these weighted representations to produce contextually enriched representations for each token. These enriched representations are subsequently fed through feedforward neural networks (i.e. an MLP) to refine their information. Finally, the output

tokens are generated based on the refined representations, and converted back to human readable format using the vocabulary (e.g. “12345+67890=80235").

3 Related Work

Interpreting and reverse engineering neural networks and transformers to find meaningful circuits has been an area of active research. Olah et al. (2020a) argued that by studying the connections between neurons and their weights, we can find meaningful algorithms (aka Circuits) in a “vision” neural network. Elhage et al. (2021) extended this approach to transformers, conceptualizing their operation in a mathematical framework that allows significant understanding of how transformer operate internally. Various tools (Foote et al., 2023; Conmy et al., 2023b) use this framework to semi-automate some aspects of reverse engineering. Nanda et al. (2023) reverse-engineered modular addition (e.g. $5 + 7 \bmod 10 = 2$) showing the model used discrete Fourier transforms and trigonometric identities to convert modular addition to rotation about a circle.

Nanda and Lieberum (2022) have argued models comprise multiple circuits. They gave examples, including the distinct training loss curve per answer digit in *5-digit* integer addition, but did not identify the underlying circuits. This work investigates and explains the circuits in **n-digit** integer addition.

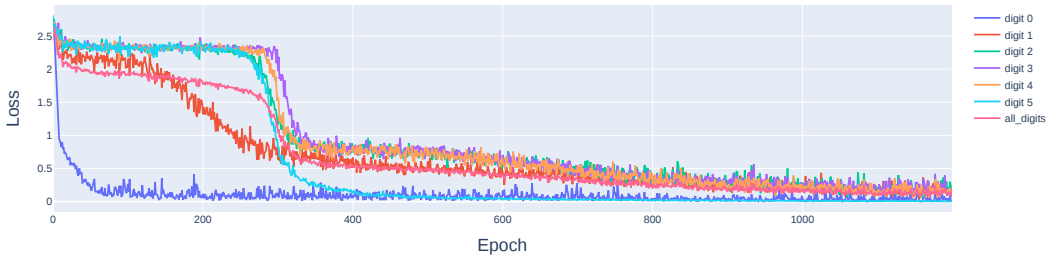

Figure 2: Training loss curves per digit position for a 5-digit integer addition task, showing the model trains each digit semi-independently.

Circuit analysis can extract graphical circuit representations and analyze component interactions (Bau et al., 2017). To enable analysis and interpretability, techniques in works like Petersen et al. (2021) symbolically reverse-engineer networks by recovering computational expressions. Research including Seth (2005) advocate analyzing networks causally, introducing structural causal models to infer mechanisms. Examinations of sequence models like Petersen et al. (2021) have analyzed the emergence and interaction of modular components during training. Evolutionary perspectives such as Miikkulainen (2021) elucidate how selection pressures shape hierarchical representations.

Information bottleneck analyses including Kawaguchi et al. (2023) relate bottlenecks to abstractions and modularization arising from forced compression. Surveys like Carbonneau et al. (2022) overview techniques to disentangle explanatory factors into separate latent dimensions. Novel objectives proposed in works like Conmy et al. (2023a) improve interpretability by encouraging modularity and disentanglement.

4 Methodology

The integer addition problem space is very dense. For 5 digit addition, there are 10 billion distinct questions (e.g. “54321+77779=") . The model must predict all 6 answer digits correctly to get the one right answer out of 200,000 possibilities. Changing a single digit in the question changes 1 to 6 digits in the answer. The full question is only revealed one token (the “=") before the model must predict the first answer digit.

Our model was trained on 1.8 million out of 10 billion questions. After training, the model predicts answers to questions with low loss, showing the model does not rely on memorisation of training data.

Fig. 2 shows the model trains each digit semi-independently suggesting the model performs integer addition by breaking down the task into parallel digit-specific streams of computation.

The traditional human addition process first sums the units before moving on to higher value digits. This is the simplest process but relies on being able to choose the order to process the digits in. This autoregressive transformer model processes text from left to right. So the model sees the higher value digits (e.g. thousands) of the question before the lower value digits (e.g. units). It doesn't use the traditional process.

A key component of addition is the need to sum each digit in the first number with the corresponding digit in the second number. Transformer models contain "attention heads" and they are the only computational sub-component of a model that can move information *between* positions (aka digits or tokens). Visualising which token(s) each attention head focussed on in each row of the calculation provided insights. While our model works with 2, 3 or 4 attention heads, 3 attention heads gives the most easily interpreted attention patterns. Fig. 3 shows the attention pattern for a single 5 digit addition calculation using 3 attention heads. Appendix C shows the same pattern for 10 and 15 digit addition. Appendix C shows the pattern with 2 or 4 attention heads.

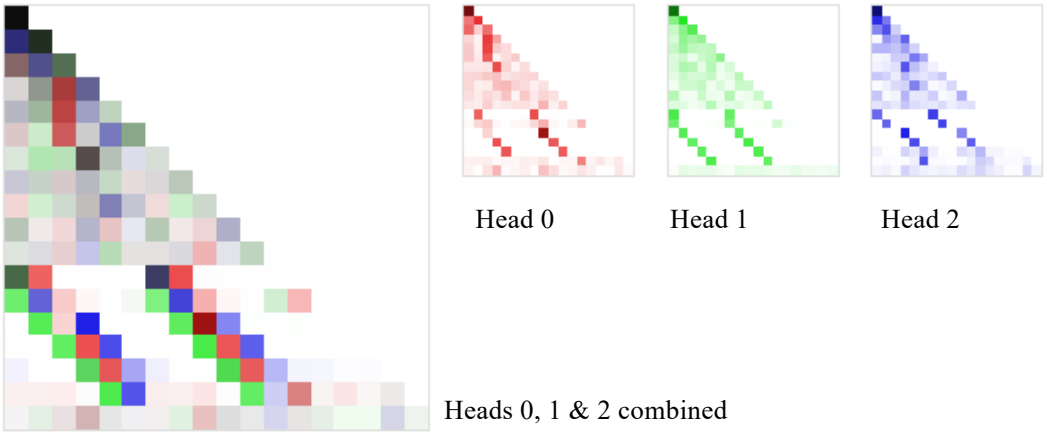

Figure 3: The attention pattern, for a model with 3 attention heads, performing a single 5 digit addition. The pattern is 18 by 18 squares (as 54321+77779=132100 is 18 tokens). Time proceeds vertically downwards, with one additional token being revealed horizontally at each row, giving the overall triangle shape. After the question is fully revealed (at row 11), each head starts attending to pairs of question digits from left to right (i.e. high-value digits before lower-value digits) giving the "double staircase" shape. The three heads attend to a given digit pair in three different rows, giving a time ordering of heads.

While it's clear the model is calculating answer digits from highest value to lowest value, using the attention heads, it's not clear what calculation each attention head is doing, or how the attention heads are composed together to perform addition.

5 Mathematical Framework

To help investigate, we created a mathematical framework describing what **any** algorithm must do if it is to perform addition correctly.

Our intuition is that the model a) incrementally discovers a necessary and sufficient set of addition sub-tasks (minimising complexity), b) discovers these sub-tasks semi-independently (maximising

parallelism), and c) treats each digit semi-independently (more parallelism). Our framework reflects this.

To explain the framework, let us first define x and y be two n-digit integers that need to be added, represented as vectors where $x = (x_0, x_1, \ldots, x_{n-1})$ and $y = (y_0, y_1, \ldots, y_{n-1})$.

We assert that the framework utilizes three base functions that operate on individual digit pairs. The first is ***Base Add*** (aka ***BA***), which calculates the sum of two digits x_i and y_i modulo 10, ignoring any carry over from previous columns. The second is ***Make Carry 1*** (aka ***MC1***), which evaluates if adding digits x_i and y_i results in a carry over of 1 to the next column. The third is ***Make Sum 9*** (aka ***MS9***), which evaluates if $x_i + y_i = 9$ exactly.

In addition, the framework uses two compound functions that chain operations across digits. The first is ***Use Carry 1*** (aka ***UC1***), which takes the previous column's carry output and adds it to the sum of the current digit pair. The second is ***Use Sum 9*** (aka ***US9***), which propagates (aka cascades) a carry over of 1 to the next column if the current column sums to 9 and the previous column generated a carry over. ***US9*** is the most complex task as it spans three digits. For some rare questions (e.g. 00555 + 00445 = 01000) ***US9*** applies to up to four sequential digits, causing a chain effect, with the ***MC1*** cascading through multiple digits. This cascade requires a time ordering of the ***US9*** calculations from lower to higher digits.

These tasks occur in the training data with different, predictable frequencies (e.g. ***BA*** is common, ***US9*** is rarer). Compound tasks are reliant on the base tasks and so discovered later in training. The discovery of each task reduces the model loss by a different, predictable amount (e.g. ***BA*** by 50%, ***US9*** by 5%). Combining these facts give an expected order of task discovery during training.

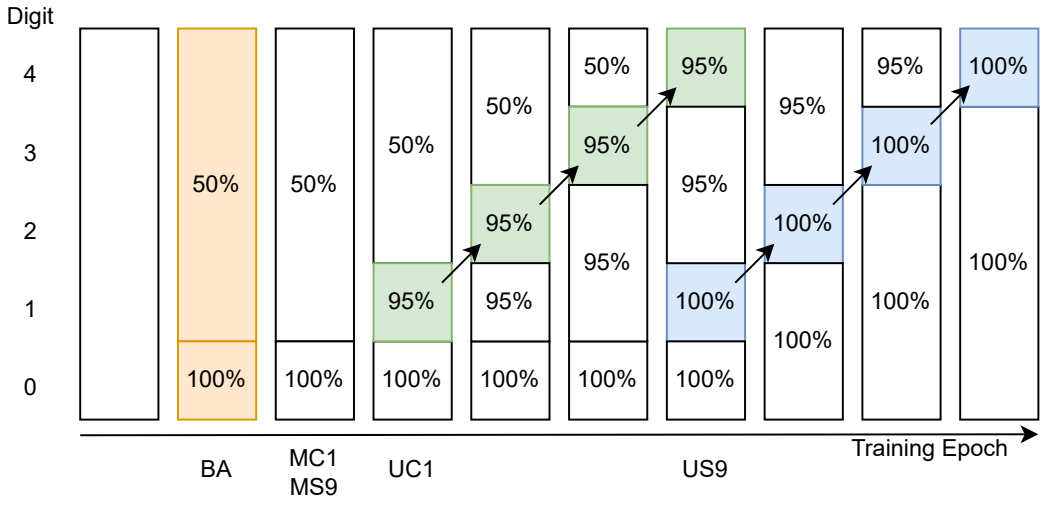

Figure 4: The mathematical framework (our method) predicts that during training, tasks are learnt for each digit independently, progressively increasing per digit accuracy (i.e. decreasing loss) shown as percentages. Mathematical rules cause dependencies between digits, giving an predicted ordering for perfect (i.e. zero loss) addition. The chain of blue squares relate to questions like 99999 + 00001 = 100000 where the ***MC1*** in digit 0 causes ***US9*** cascades through multiple other digits.

We use this mathematical framework solely for analysis to gain insights. The model training and all loss calculations are completely independent of this mathematical framework.

6 Training Analysis

Fig. 2 shows the model trains each digit semi-independently. Armed with the mathematical framework, we investigated each digit separately.

The Digit 0 calculation is the least interesting as it only uses ***BA*** (not ***UC1*** or ***US9***). Once discovered, Digit 0 always quickly refines to have the lowest loss and least noise (as expected). (Graphs in Appendix B.)

For the other digits, we categorised the training data into 3 non-overlapping subsets aligned to the ***BA*** , ***UC1*** and ***US9*** tasks, and graphed various combinations, finding interesting results.

The ***US9*** graphs are much noisier than other graphs (Fig. 5). We found that the model has low loss on simple ***US9*** cases (e.g. 45 + 55 = 100) but has high loss on ***US9*** cascades (e.g. 445 + 555 = 1000) where the ***MC1*** must be propagated "right to left" two 2, 3 or 4 columns. The model can't perform these rare use cases safely, as it has a "left to right" algorithm.

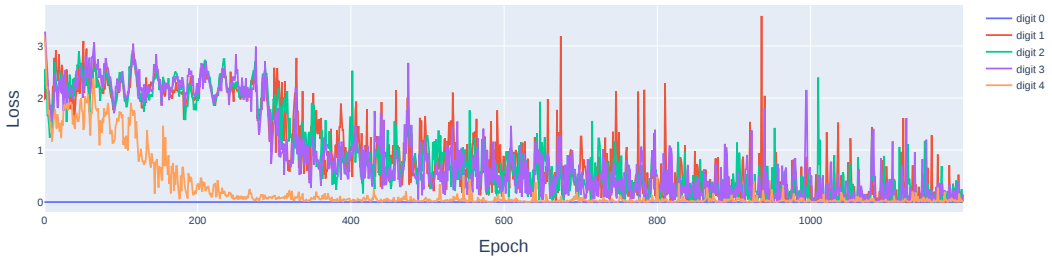

Figure 5: High variability in the per digit training loss for ***US9*** cases caused by the model's inability to reliably do cascading ***US9*** cases such as 445+555=1000.

Graphing the ***BA*** and ***UC1*** use cases side by side for any one of the Digits 1, 2 and 3 shows an interesting pattern (Fig. 6). In Phase 1, both tasks have the same (high) loss. In Phase 2, both curves drop quickly but the ***BA*** curve drops faster than the ***UC1*** curve. This "time lag" matches our expectation that the ***BA*** task must be accurate before the ***UC1*** task can be accurate. In Phase 3, both tasks' loss curve decrease slowly over time.

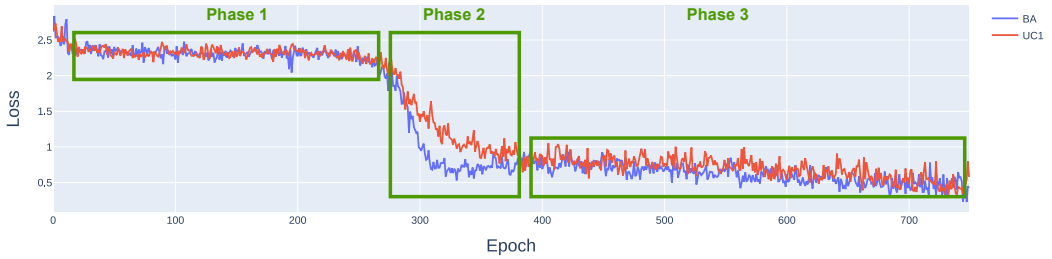

Figure 6: Training loss for digit 3 showing that, in Phase 2, the refining of ***Use Carry 1*** lags behind ***Base Add*** . ***Base Add*** and ***Use Carry 1*** are refined separately and have separate calculation algorithms. The 3 phases seem to correspond to "memorisation", "algorithm discovery" and "clean-up".

Both the ***BA*** and ***UC1*** tasks need to move data between tokens, and so will be implemented in attention head(s). Fig. 6 shows they are trained semi-independently. We choose the number of attention heads in our model with the clearest separation of tasks in the attention pattern. We find (later) that our model has separate attention heads for the ***BA*** and ***UC1*** tasks.

Digit 4, the highest question digit, has a significantly different loss curve (shown in Fig. 7) than Digits 1, 2 and 3. This is partially explained by Digit 4 only having simple ***US9*** cases (i.e. no ***US9*** cascades). This does not explain the ***BA*** or ***UC1*** differences. This difference persists with different seed values, and with 10 or 15 digit addition. We explain this difference later.

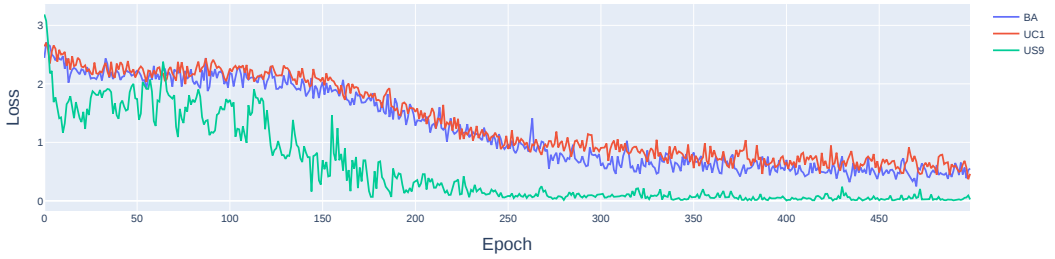

Figure 7: Training loss for digit 4 starts and stays lower for all tasks than it does for digits 1, 2 and 3. Digit 4 has a different calculation algorithm from digits 1, 2 and 3.

7 Prediction Analysis

During model prediction, we overrode (mean ablated) the model memory (residual stream) at each row, and confirmed that the addition algorithm does **not** use any data generated in rows 0 to 10 inclusive. In these rows the model has **not** yet seen the full question and every digit in the question is independent of every other digit, making accurate answer prediction infeasible. The model also does not use the last (17th) row. Therefore, the addition is started and completed in 6 rows (11 to 16). Further (ablation) experiments confirmed that the A0 to A4 answers are calculated one row before being revealed. (Details in Appendix H.)

The model has slightly different algorithms for the first digit pairs, the middle digit pairs and the last digit pairs. Fig. 1 has a simplified version of how the model calculates the middle digit pair A3. Fig. 8 has more details. For 5 digit addition, there are 2 middle digit pairs (A3 and A2) whereas for 15 digit addition there are 12 middle digit pairs.

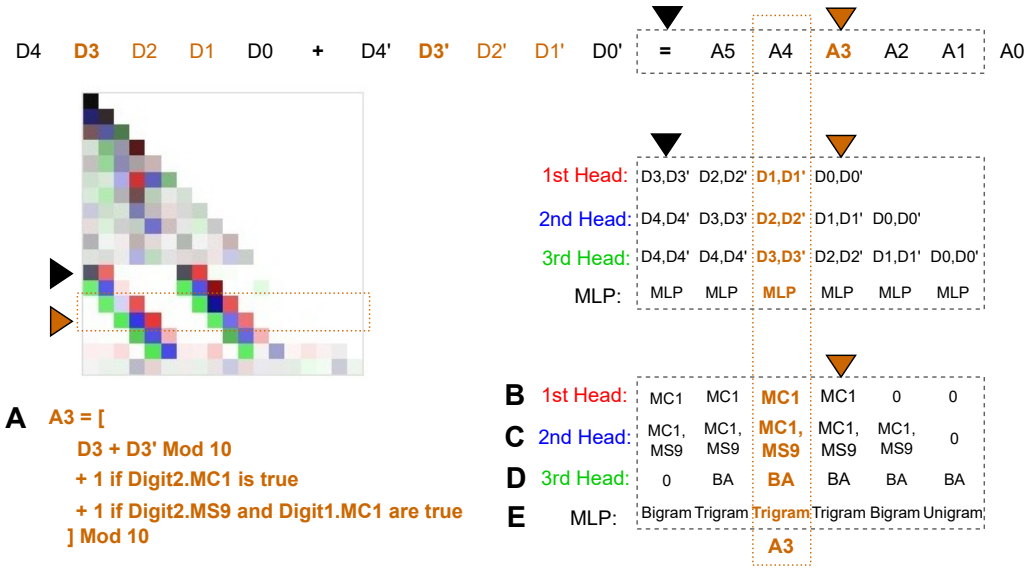

Figure 8: **A:** For A3, the addition algorithm must combine information from digits 3, 2 and 1. **B:** The 1st head calculates ***MC1*** on digit 1. **C:** The 2nd head calculates ***MC1*** and ***MS9*** (at most one of which can be true) on digit 2. **D:** The 3rd head calculates ***Base Add*** on digit 3. **E:** The MLP layer uses trigrams to combine the information from the 3 heads to give the final answer A3, one row before it is output. Appendix G shows this algorithm as pseudocode.

The A3 addition algorithm has three clauses related to digits 3, 2 and 1. Ablating each head in turn shows that the 3rd head has most impact on loss, the 2nd head has less impact, and the 1st head has

little impact. This aligns with the intuition that the sum "D3 + D3'" matters most, the ***MC1*** from the previous digit (D2 + D2') matters less, and the rare ***MC1*** from the previous previous digit (D1 + D1') matters least.

The last two digits, A1 and A0, use a simplified a version of the A3 algorithm, with some of the three clauses not needed.

The A3 algorithm could also successfully be applied to A4. But the Digit 4 training curve is better (faster) than the middle digits. The attention patterns shows that for A4, the model is using all the heads in row 11 (the "=" token) when the A3 algorithm doesn't require this. Uniquely, A4 utilises more "compute" than is available to A3, A2, A1 or A0. We assume the model uses this advantage to implement a faster-training and lower-loss algorithm for A5 and A4. We haven't worked out the details of this.

Mean ablating the 1st or 2nd head slightly increased the average loss for ***BA*** questions from 0.05 to 0.08, whereas ablating the 3rd head substantially increased the loss to 3.7, confirming that the 3rd head is doing the ***BA*** task. (Details in Appendix H.)

The MLP can be thought of as a "key-value pair" memory (Meng et al., 2022; Geva et al., 2021) that can hold many bigrams and trigrams. We claim our MLP pulls together the two-state 1st head result, the tri-state 2nd head result and the ten-state 3rd head result value, treating them as a trigram with 60 (2 x 3 x 10) possible keys. For each digit, the MLP has memorised the mapping of these 60 keys to the 60 correct digit answers (0 to 9). We haven't proven this experimentally. Our MLP is sufficiently large to store this many mappings with zero interference between mappings (Elhage et al., 2022).

Despite being feasible, the model does **not** calculate the task ***MC1*** in rows 7 to 11. Instead it completes each digit calculation in 1 row, possibly because there are training optimisation benefits in generating a "compact" algorithm.

This algorithm explains all the observed prediction behaviour - including the fact that the model can calculate a simple ***US9*** case but not a cascading ***US9*** case. We assume that, given the dense nature of the question and answer, and the small model size, the model does not have sufficient time and compute resources to implement both ***UC1*** and ***US9*** accurately, and so preferences implementing the more common (***UC1***) case, and only partially implements the more complex and rare (***US9***) case.

8 Conclusions and Future Work

This work demonstrates a successful approach to reverse engineering and elucidating the emergent algorithm within a transformer model trained on integer addition. By combining mathematical analysis, empirical investigation of training and prediction, and targeted experimentation, we are able to explain how the model divides the task into parallel digit-specific streams, employs distinct subroutines for different digit positions, postpones calculations until the last possible moment yet executes them rapidly, and struggles with a specific rare case.

Our theoretical framework of necessary addition subtasks provides a foundation for the model's behavior. The digit-wise training loss curves reveal independent refinement consistent with separate digit-specific circuits. Attention patterns illustrate staging and time-ordering of operations. Controlled ablation experiments validate ouur hypothesis about algorithmic elements' roles. Together these methods enable a detailed accounting of the model's addition procedure.

This methodology for mechanistic interpretability, when applied to broader tasks and larger models, can offer insights into not just what computations occur inside complex neural networks, but how and why those computations arise. Such elucidation will be increasingly important for ensuring the safety, reliability and transparency of AI systems.

Our study paves the way for numerous potential research avenues. Recognizing the challenges in rare cases can inspire methods to enhance the robustness of addition models. The established framework might be adapted to elucidate models for integer subtraction or multiplication. By integrating proven and effective addition modules into a larger, untrained network geared towards multiplication, the training process could be expedited. Further, decoding the multiplication algorithm becomes more straightforward when the addition-related tasks are already recognized and deemed reliable. Utilizing this modular approach can simplify the understanding of intricate algorithms, propelling

advancements in the field of mechanistic interpretability. In summary, this research underscores that diving deep into the workings of contemporary machine learning can highlight valuable strengths, pinpoint areas for improvement, and present avenues for accelerated progress.

9 Reproducibility Statement

To ensure our work is reproducible, we provide the full source code in the supplementary materials, as well as all necessary data and parameters and instructions to reproduce our experimental results.

10 Acknowledgements

We are thankful to Jason Hoelscher-Obermaier for his comments on the earlier draft, Esben Kran for organising the Interpretability Hackathon and Clement Neo for his comments on the paper and assistance with some of the figures. This project was supported by Apart Lab.

References

F. Barez, H. Hasanbieg, and A. Abbate. System iii: Learning with domain knowledge for safety constraints, 2023.

D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network dissection: Quantifying interpretability of deep visual representations. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3319–3327, 2017.

M.-A. Carbonneau, J. Zaidi, J. Boilard, and G. Gagnon. Measuring disentanglement: A review of metrics, 2022.

A. Conmy, A. Mavor-Parker, A. Lynch, S. Heimersheim, and A. Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability, 04 2023a.

A. Conmy, A. N. Mavor-Parker, A. Lynch, S. Heimersheim, and A. Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability, 2023b.

N. Elhage, N. Nanda, C. Olsson, T. Henighan, N. Joseph, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, N. DasSarma, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah. A mathematical framework for transformer circuits. https://transformer-circuits.pub/2021/framework/index.html, 2021.

N. Elhage, T. Hume, C. Olsson, N. Schiefer, T. Henighan, S. Kravec, Z. Hatfield-Dodds, R. Lasenby, D. Drain, C. Chen, R. Grosse, S. McCandlish, J. Kaplan, D. Amodei, M. Wattenberg, and C. Olah. Toy models of superposition, 2022.

A. Foote, N. Nanda, E. Kran, I. Konstas, S. Cohen, and F. Barez. Neuron to graph: Interpreting language model neurons at scale, 2023.

M. Geva, R. Schuster, J. Berant, and O. Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.446. URL https://aclanthology.org/2021.emnlp-main.446.

K. Kawaguchi, Z. Deng, X. Ji, and J. Huang. How does information bottleneck help deep learning?, 2023.

K. Meng, D. Bau, A. Andonian, and Y. Belinkov. Locating and editing factual associations in gpt. https://proceedings.neurips.cc/paper_files/paper/2022/file/6f1d43d5a82a37e89b0665b33bf3a182-Paper-Conference.pdf, 2022.

R. Miikkulainen. Creative ai through evolutionary computation: Principles and examples. https://doi.org/10.1007/s42979-021-00540-9, 2021.

N. Nanda and T. Lieberum. Mechanistic interpretability analysis of grokking. https://www.alignmentforum.org/posts/N6WM6hs7RQMKDhYjB, 2022.

N. Nanda, L. Chan, T. Lieberum, J. Smith, and J. Steinhardt. Progress measures for grokking via mechanistic interpretability, 2023.

C. Olah, N. Cammarata, L. Schubert, G. Goh, M. Petrov, and S. Carter. Zoom in: An introduction to circuits. https://distill.pub/2020/circuits/zoom-in/, 2020a.

C. Olah, N. Cammarata, L. Schubert, G. Goh, M. Petrov, and S. Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024.001, 2020b.

A. Petersen, P. Voigtlaender, and A. Krause. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *Advances in Neural Information Processing Systems*, 34, 2021.

T. Räuker, A. Ho, S. Casper, and D. Hadfield-Menell. Toward transparent ai: A survey on interpreting the inner structures of deep neural networks. *arXiv preprint arXiv:2207.13243*, 2022.

A. K. Seth. Causal connectivity of evolved neural networks during behavior. *Network: Computation in Neural Systems*, 16(1):35–54, 2005. doi: 10.1080/09548980500238756. URL https://doi.org/10.1080/09548980500238756.

J. Vig. A multiscale visualization of attention in the transformer model. *arXiv preprint arXiv:1906.05714*, 2019.

A Appendix - Number of Attention Heads

The model can be successfully trained with 2, 3 or 4 attention heads. However as described in (Fig. 10), 3 attention heads is more useful for human understanding than 2 or 4 heads.

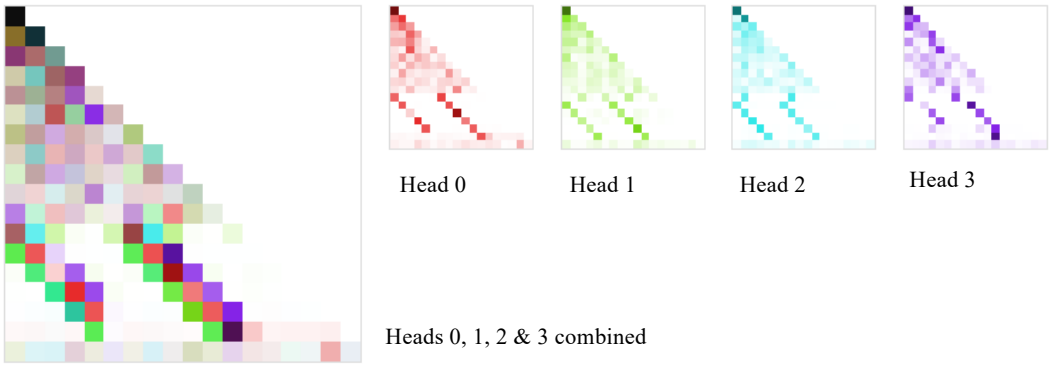

Figure 9: For 5 digit addition, using 4 attention heads gives an attention pattern where the H1 and H2 staircases overlap perfectly. Ablating one of H1 or H2 increases loss. The similarity in H1 and H2's attention patterns suggests they are "splitting" a single logical task. Spliting is feasible as Elhage et al. (2021) says attention heads are independent and additive.

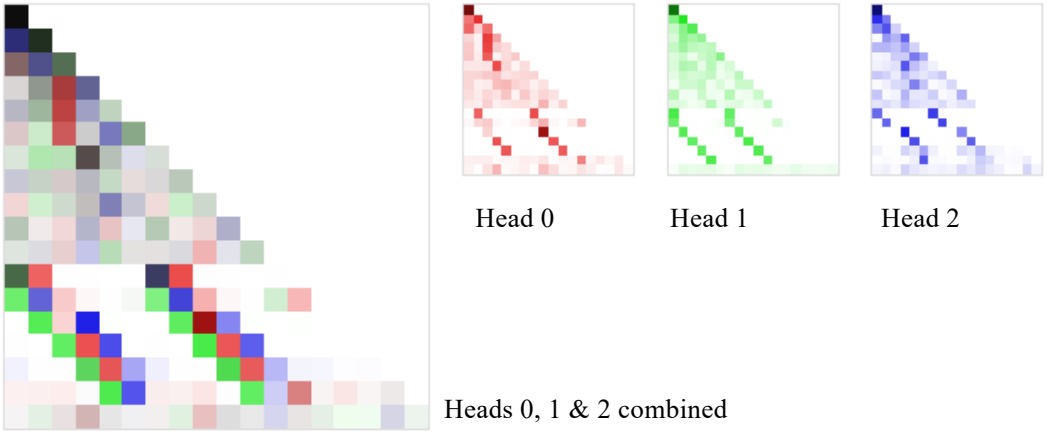

Figure 10: For 5 digit addition, using 3 attention heads gives the best separation with the heads having distinct, non-overlapping attention pattern. Ablating any head increases the model loss, showing all 3 heads are useful.

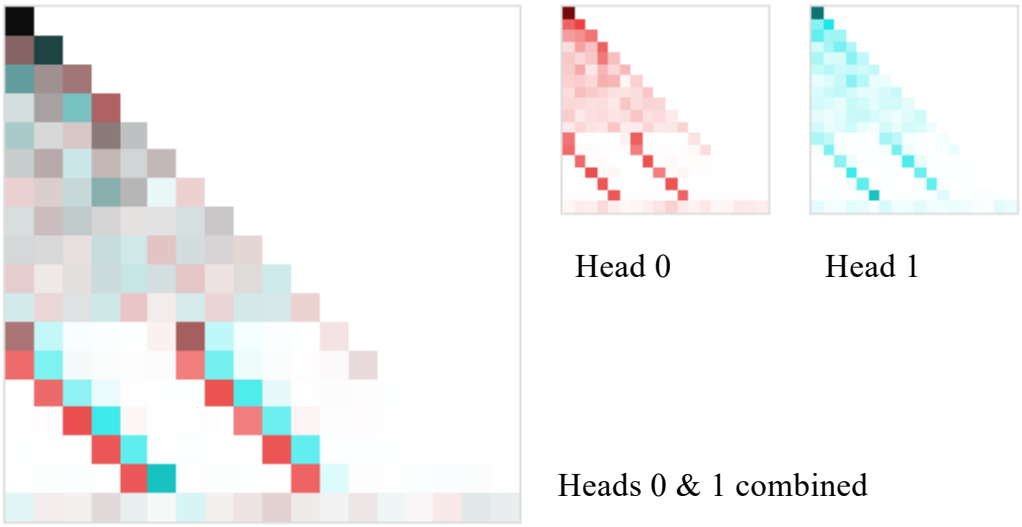

Figure 11: For 5 digit addition, using 2 attention heads works, but the model attends to multiple token pairs in a single head, suggesting that multiple tasks are being packed into a single head, which makes it harder to understand.

B Appendix - Training Loss by Number of Digits

The model can be successfully trained with 2, 5, 10, 15, etc digits. As expected, for a given loss threshhold, 15 digit addition takes longer to train than 10 digit addition, which takes longer to train than 5 digit addition.

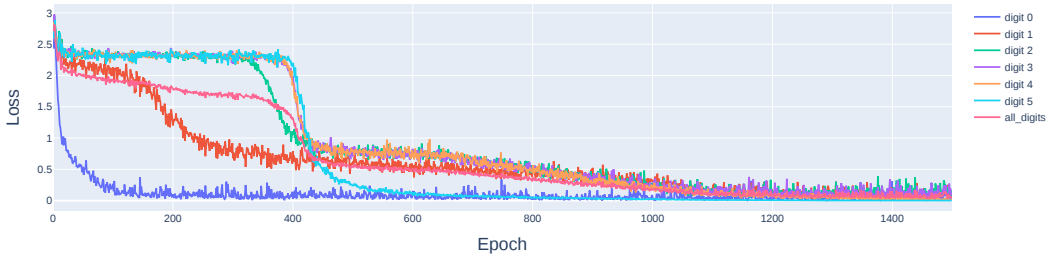

Figure 12: The per-digit training loss curves for 5 digit addition

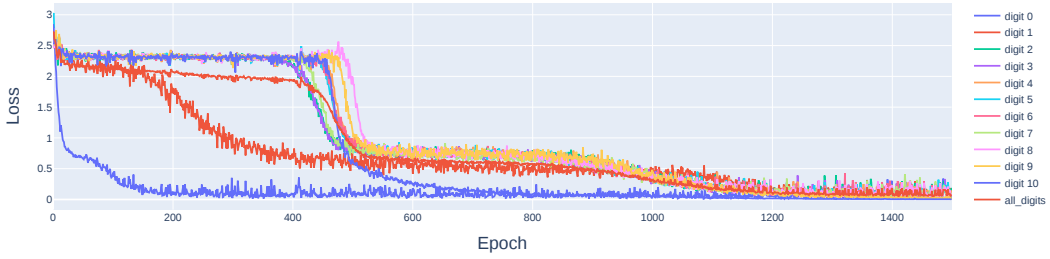

Figure 13: The per-digit training loss curves for 10 digit addition.

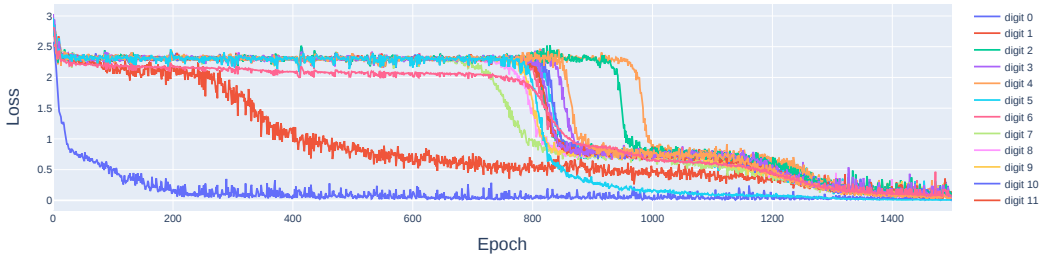

Figure 14: The per-digit training loss curves for 15 digit addition.

C Appendix - Attention Patterns by Number of Digits

The model can be successfully trained with 2, 5, 10, 15, etc digits. The attention patterns for theses cases show similarities that aid in human understanding.

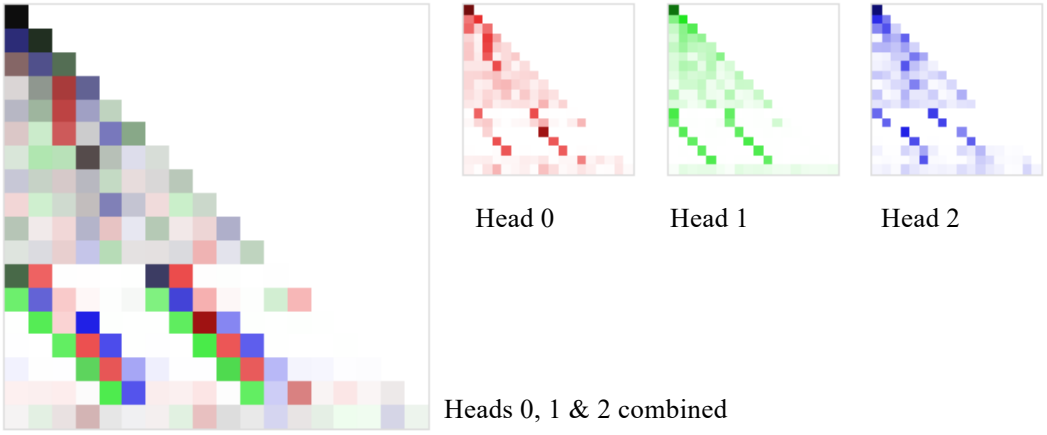

Figure 15: For 5 digit addition, and 3 attention heads, the attention pattern has a strong double-staircase shape. Each step of the staircase is 3 blocks wide, showing the heads are attending to different tokens in each row.

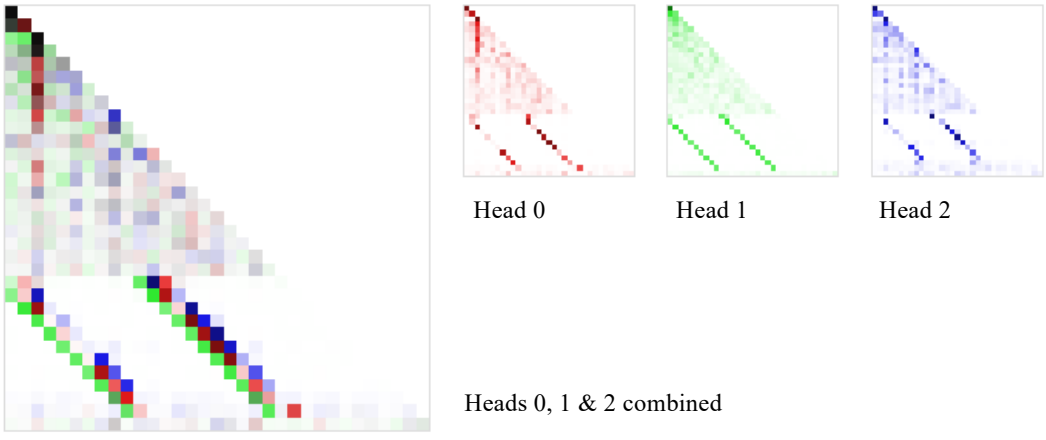

Figure 16: For 10 digit addition, and 3 attention heads, the attention pattern has a strong double-staircase shape. Each step of the staircase is 3 blocks wide, showing the heads are attending to different tokens in each row.

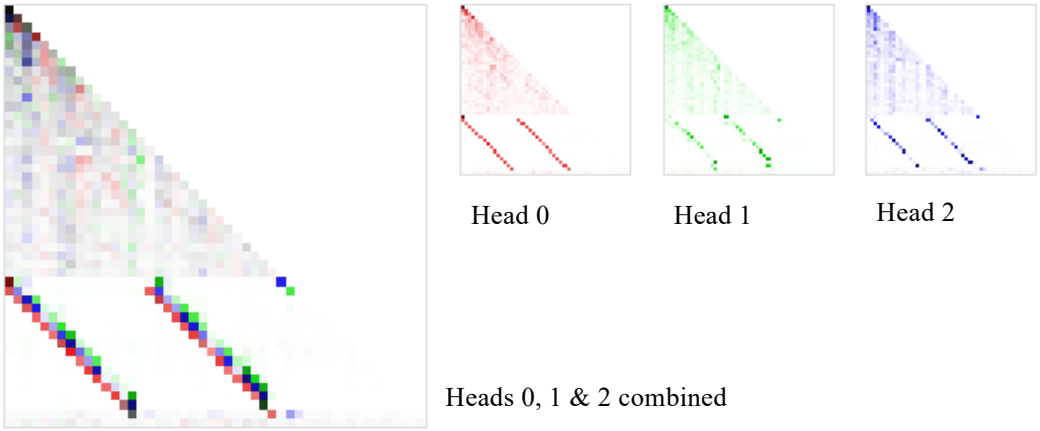

Figure 17: For 15 digit addition, and 3 attention heads, the attention pattern has a strong double-staircase shape. Each step of the staircase is 3 blocks wide, showing the heads are attending to different tokens in each row.

D Appendix - Model Configuration

Experimentation was done in a CoLab notebook:

- It will be loaded to github post publication.
- It runs on a T4 GPU with each experiment taking a few minutes to run.
- The key parameters (which can all be experimented with) are:
  1. n_layers = 1; This is a one layer Transformer
  2. n_heads = 3; There are 3 attention heads
  3. n_digits = 5; Number of digits in the addition question
- It uses a new batch of data each training step (aka Infinite Training Data) to minimise memorisation.
- During a training run the model processes about 1.5 million training datums. For the 5 digit addition problem there are 100,000 squared (that is 10 billion) possible questions. So the training data is much less than 1% of the possible problems.
- Because ***US9*** cascades (e.g. 44445+55555=100000, 54321+45679=1000000, 44450+55550=10000, 1234+8769=10003) are exceedingly rare, the data generator was enhanced to increase the likelihood of these cases turning up in the training data.

The CoLab notebook can be downloaded from https://github.com/apartresearch/conceptual-interp

E Appendix - Solving a jigsaw using many people

Here is useful analogy for changing how we approach a problem to more closely resemble how a Transformer would; a jigsaw puzzle. A single person could solve it using a combination of meta knowledge of the problem (placing edge pieces first), categorisation of resources (putting like-coloured pieces into piles), and an understanding of the expected outcome (looking at the picture on the box).

But if instead we had one person for each piece in the puzzle, who only knew their piece, and could only place it once another piece that it fit had already been placed, but couldn't talk to the other people, and did not know the expected overall picture, the strategy for solving the jigsaw changes dramatically.

When they start solving the jigsaw, the 4 people holding corner pieces place them. Then 8 people holding corner-adjacent edge pieces can place them. The process continues, until the last piece is placed near the middle of the jigsaw.

We posit that this approach parallels how transformer models work. There is no pre-agreed overall strategy or communication or co-ordination between people (circuits) - just some “rules of the game" to obey. The people think independently and take actions in parallel. The tasks are implicitly time ordered by the game rules.

F Appendix - *Use Sum 9* Training Loss Graph

Fig. 18 shows a training loss curve for just ***BA*** and ***UC1*** . The model has high loss on ***Use Sum 9*** cascades shown as high variability in the loss graph (See Fig. 19).

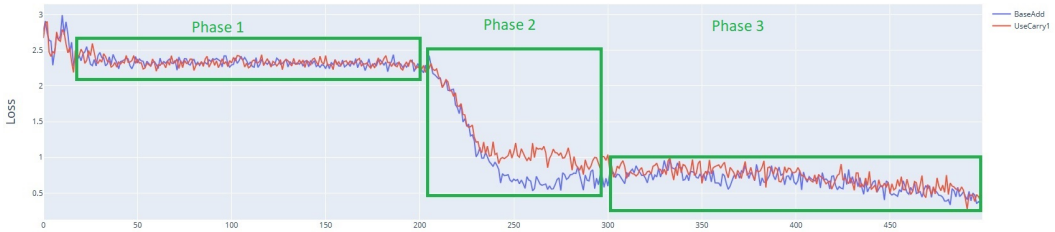

Figure 18: Training loss for digit 3 showing that, in Phase 2, the refining of ***Use Carry 1*** lags behind ***Base Add*** . This supports the claim that ***Base Add*** and ***Use Carry 1*** are refined separately and have separate calculation algorithms.

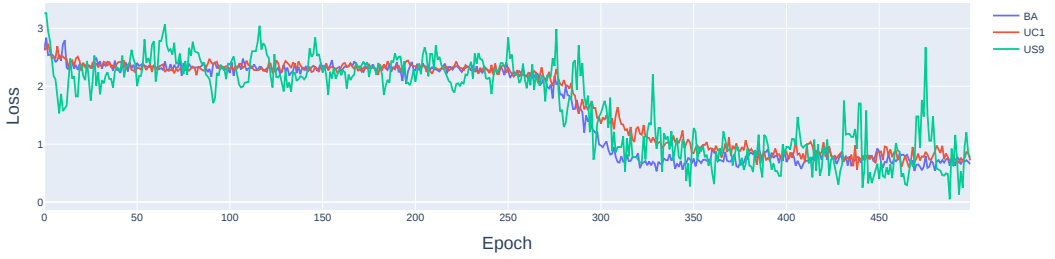

Figure 19: Training loss for digit 3 for the ***Base Add*** , ***Use Carry 1*** and ***Use Sum 9*** tasks, showing the model struggles to reduce loss on ***Use Sum 9*** compared to ***Base Add*** and ***Use Carry 1*** .

G Appendix - Model Algorithm as Pseudocode

This section reproduces the algorithm summarised in Fig. 8 as pseudocode. This code calculates the largest digit first, handling simple (but not cascading) UseSum9 cases, and returning the answer.

Note that the pseudocode does not retain any information between passes through the for loop - this corresponds to each digit being calculated independently of the other digits.

Algorithm 1 n-digit integer addition algorithm

function CALCULATEANSWERDIGITS($n\ digits, q1, q2$)
  answer $\leftarrow 0$ ▷ Initialize the answer to zero
  for all $i = 0, \ldots, n\ digits - 1$ **do** ▷ Loop over each digit
    pos $\leftarrow n\ digits - i - 1$ ▷ Current position from the right
    prev pos $\leftarrow$ pos $- 1$ ▷ Previous position from the right
    prev prev pos $\leftarrow$ pos $- 2$ ▷ Position before the previous
    mc1 prev prev $\leftarrow 0$ ▷ Initialize carry from two positions before
    if prev prev pos ≥ 0 **then**
      if $q1_{\text{prev prev pos}} + q2_{\text{prev prev pos}} \geq 10$ **then**
        mc1 prev prev $\leftarrow 1$ ▷ Calculate carry from two positions before
      end if
    end if
    mc1 prev $\leftarrow 0$ ▷ Initialize carry from previous position
    if prev pos ≥ 0 **then**
      if $q1_{\text{prev pos}} + q2_{\text{prev pos}} \geq 10$ **then**
        mc1 prev $\leftarrow 1$ ▷ Calculate carry from the previous position
      end if
    end if
    ms9 prev $\leftarrow 0$ ▷ Initialize a flag for sum of 9 in previous position
    if prev pos ≥ 0 **then**
      if $q1_{\text{prev pos}} + q2_{\text{prev pos}} == 9$ **then**
        ms9 prev $\leftarrow 1$ ▷ The previous position's sum is 9
      end if
    end if
    prev prev $\leftarrow 0$ ▷ Initialize carry adjustment for a sum of 9
    if mc1 prev $== 0$ **then**
      if ms9 prev $== 1$ **then**
        if mc1 prev prev $== 1$ **then**
          prev prev $\leftarrow 1$ ▷ Adjust carry if there's a 9 and carry from 2 positions before
        end if
      end if
    end if
    digitanswer $\leftarrow q1_{\text{pos}} + q2_{\text{pos}} +$ mc1 prev $+$ prev prev ▷ Calculate the answer for the current digit
    digitanswer $\leftarrow$ MODULUS(digitanswer, 10) ▷ Correct the current digit if it's ≥ 10
    answer $\leftarrow$ digitanswer $+$ answer $\times 10$ ▷ Concatenate the current digit to the answer
  end for
    return answer ▷ Return the final calculated answer
end function

H APPENDIX - THE LOSS FUNCTION AND LOSS MEASURES

The loss function is simple:

- Per Digit Loss: For "per digit" graphs and analysis, for a given answer digit, the loss used is negative log likelihood.
- All Digits Loss: For "all answer digits" graphs and analysis, the loss used is the mean of the "per digit" loss across all the answer digits.

The final training loss varies with the number of digits in the question as shown in Tab. 1.

Size of question	Final training loss	Example question
5 digit addition	0.009	11111 + 22222 = 033333
10 digit addition	0.011	1111111111 + 2222222222 = 03333333333
15 digit addition	0.031	111111111111111 + 222222222222222 = 0333333333333333
Overall Average	**0.017**	**General Performance**

Table 1: The final training loss after 5000 training epochs, each containing 64 questions, using All Digits Loss, for different size addition models.

The final training loss for each digit in the question varies as shown in Tab. 2.

Digit index	Training loss	AKA	Example of digit
5	<0.001	A5	11111+22222=**0**33333
4	<0.001	A4	11111+22222=0**3**3333
3	0.003	A3	11111+22222=03**3**333
2	0.008	A2	11111+22222=033**3**33
1	0.046	A1	11111+22222=0333**3**3
0	0.001	A0	11111+22222=033333
Overall Average	**0.010**		

Table 2: For 5-digit addition, final training loss per digit.

Using our mathematics framework, we categorize each training question as one of following:

- ***BA*** : BaseAdd questions only require "base add" calculations to get the correct answer.
- ***MC1*** : MakeCarry1 questions require "use carry 1" and "base add" calculations to get the correct answer.
- ***US9*** : UseSum9 questions require "use sum 9", "use carry 1" and "base add" calculations to get the correct answer.

This allows us to measure the loss per category as shown in Tab. 3. The frequency of these question types differ significantly (***BA*** = 61%, ***MC1*** =33%, ***US9*** =6%) so the final training loss values, at this level, are not very informative.

Question type	Training loss	Aka	Example of digit
BA	0.021	Base Add	11111+22222=033333
MC1	0.001	Make Carry 1	11811+22222=034033
US9	<0.001	Use Sum 9	17811+22222=040033
Overall Average	**0.011**	**General Performance**	

Table 3: For 5 digit addition, the final training loss per question category.

After training, we used the model to give answers to questions. To understand whether the calculations at token n are important, we look at the impact on loss of ablating all attention heads at that token (using the TransformerLens framework, zero ablation of the blocks.0.hook_resid_post data set, the above loss function, and a "cut off" loss threshold of 0.08). Tab. 4 shows sample results.

Token	Average loss	Conclusion
0 .. 10	0.070	Impact is low. Calcs for these 11 tokens are unimportant
11	0.322	Loss is 4 x threshold. Calculations are important
12	0.497	Loss is 6 x threshold. Calculations are important
13	0.604	Loss is 7 x threshold. Calculations are important
14	0.921	Loss is 11 x threshold. Calculations are important
15	1.181	Loss is 14 x threshold. Calculations are important
16	1.021	Loss is 12 x threshold. Calculations are important
17	0.070	Impact is low. Calculations for this token is unimportant

Table 4: For 5 digit addition, for the 18 tokens / calculation rows, how abalting each token/row impacts the calculation loss.

This shows that all important calculations are completed while the model processes just 6 of the 18 tokens.

For deeper investigation, we created $\tilde{1}00$ hand-curated test questions. They cover all question types (BA, MC1 and MS9) and all the answer digits (A5 .. A0) that these question types can occur in. These test cases were not used in model training. Example questions include:

- make_a_question(_, _, 888, 11111, BASE_ADD_CASE)
- make_a_question(_, _, 35000, 35000, USE_CARRY_1_CASE)
- make_a_question(_, _, 15020, 45091, USE_CARRY_1_CASE)
- make_a_question(_, _, 25, 79, SIMPLE_US9_CASE)
- make_a_question(_, _, 41127, 10880, SIMPLE_US9_CASE)
- make_a_question(_, _, 123, 877, CASCADE_US9_CASE)
- make_a_question(_, _, 81818, 18182, CASCADE_US9_CASE)

The above experiment was repeated (with "cut off" loss threshold of 0.1) using these $\tilde{1}00$ questions (and another 64 random questions). We analyzed the incorrect answers, grouping these failures by which answer digits were wrong. Tab. 5 shows sample results.

Row	Number of incorrect answers, grouped by incorrect digits
11	**'Nyyyyy': 47**, 'NyNyyy': 5, 'yNyyyy': 7, 'yyNyyy': 7, 'yyyNyy': 1, ...
12	**'yNyyyy': 97**, 'yNNyyy': 7, 'NNyyyy': 3, 'Nyyyyy': 2, 'yyyNyy': 1, ...
13	**'yyNyyy': 85**, 'NyNyyy': 3, 'yNNyyy': 2, 'Nyyyyy': 3, 'yNyyyy': 5, ...
14	**'yyyNyy': 72**, 'yyNNyy': 4, 'NyyNyy': 3, 'yNyNyy': 3, 'Nyyyyy': 2, ...
15	**'yyyyNy': 74**, 'yyNyNy': 3, 'Nyyyyy': 2, 'NyyyNy': 3, 'yNyyNy': 3, ...
16	**'yyyyyN': 82**, 'yyNyyN': 2, 'NyyyyN': 4, 'yNyyyN': 4, 'yyNyyy': 9, ...

Table 5: For 5 digit addition, how ablating all the attention heads for a row impacts answer digit correctness. An 'N' means that answer digit was incorrect in the predicted answer. In each step, the failure count for the top grouping is > 6 times the next most common grouping - evidence that each step calculates one answer digit. There is a very steady progression through the rows with each successive row focusing on the next answer digit - this progression reflects numerically the "double staircase" attention pattern.

We also ablated one head at a time (using the TransformerLens framework, mean ablation of the blocks.0.attn.hook_z data set, the standard loss function, and a "cut off" loss threshold of 0.08) to understand which head(s) were key in calculating BA questions. Tab. 6 shows sample results.

The last row of Tab. 6 shows that digit A0 was calculated in step 16. This is one step before the model reveals A0 in step 17. The other rows in the table shows that this pattern holds true for all the other digits: each digit is calculated one step before the model reveals it.

Ablated Head	Average loss	Conclusion
0	0.016	Impact is low. Head is unimportant for ***BA***
1	4.712	Loss is 50 x threshold. Head is important for ***BA***
2	0.062	Impact is low. Head is unimportant for ***BA***

Table 6: For 5 digit addition, when ablating heads, Head 1 is clearly key for the calculation of ***Base Add*** questions.